FAULT-TOLERANT EMBEDDED COMPUTER SYSTEM-ON-CHIP FOR ENDOSCOPE CONTROL

Gul N. KHAN
Department of Electrical and Computer Engineering
Ryerson University, 350 Victoria Street
Toronto, Ontario M5B 2K3 Canada

Siew Hoon WEE
Innomedia Pte Ltd
10 Science Park Road, #03-04
Singapore Science Park II, Singapore 117684

Abstract: The architecture of a fault tolerant embedded computer system is presented, which implements a machine vision based navigation system of a colonoscope. The automation of colonoscopy requires high performance computing for image analysis and navigation while fault-tolerance for patient safety. High performance computing and real-time control is achieved by a hardware engine of heterogeneous multiple processors. The software architecture consists of system as well as application level endoscope control processes. The target architecture adapts itself to the varying needs of machine vision and endoscope navigation algorithms. Embedded system development is a hardware-software co-design problem where the hardware and software architectures are designed simultaneously. The virtual system hardware is constructed using HDL-based virtual software processors and hardware modules. The system and application software is developed and co-verified with the virtual hardware using Eaglei toolset. The hardware software co-verification results indicate that the system performance degrades gracefully under various fault scenarios.

1. INTRODUCTION

Embedded computer systems are built to constantly respond to external events and to generate control signals as function of their current state and inputs from sensors. The essential features of safety-critical application embedded systems are high reliability, fault-tolerance and adaptability of the system. We present the architecture of a fault-tolerant embedded computer system being employed to implement a vision-based navigation system of an endoscope. Endoscope is a medical instrument used for diagnosing UGI, colon and bronchus diseases. We are concentrating on the computerization of colonoscopes. The colonoscopy automation requires high performance computation for machine vision and navigation algorithms and fault-tolerance for patient safety. The high performance computation and real-time interfacing is achieved by a hardware engine of heterogeneous multiple processors. The software architecture consists of allocation and scheduling of processes, support for fault-tolerance, inter-process communication and application processes. The aim of this project is to design a high performance embedded computer system to implement machine vision algorithms, build a search space representation of colon and control the endoscope tip. Standard industrial and parallel computer systems have been employed in the past [1]. However, it is observed that a dedicated and fault-tolerant embedded system is needed for such safety-critical applications.

2. EMBEDDED COMPUTER ARCHITECTURE

A block diagram of the embedded system and its interface with a typical colonoscope is shown in Figure 1. The system consists of five processors, which are fully connected by using ten dual port memories (**DP$_{ij}$**) as shown in Figure 2. It has three compute processors (**WP**) for computational intensive tasks and two interface processors (**IP**) for real-time control and I/O. The processor interconnection network is made of dual port, **DP** memories and it supports high-speed message transfer. It also provides alternate routes for inter-processor communication in case a particular **DP** memory or its interface fails. **DP** memories or their interface failure is detected during message transfer. A watchdog timer detects the node failure. The **IP** processor provides fault tolerant support in addition to performing task scheduling and load balancing. One of the **IP** processors serves as system controller to monitor and isolate faulty components while the other **IP** can work as backup controller by monitoring the designated controller. The processor architecture supports various software and hardware fault-tolerance strategies. Three compute processors can be connected in TMR (triple modular redundant) configuration where one of the **IP** processor serves as a voter. The architecture is flexible and both **IP** processors can be configured as a duplex voter for the TMR arrangement.

The embedded computer system can be envisaged as having computing and IO node partitions. The IP nodes are critical from the endoscope control point of view. These nodes interface with the sensors and issues tip movement commands. The ADC module digitizes the stream of colon image, which are passed to computing nodes that execute machine vision algorithms and build a search space representation of the colon. Digital outputs are used to control the endoscope tip direction and a couple of standard analog input channels fed back the actual tip movements. The endoscope tip control commands are generated in the form of digital pulses that drives two DC motors. These motors direct the tip in two concentric perpendicular directions by using the same control wires employed by manual control wheels.

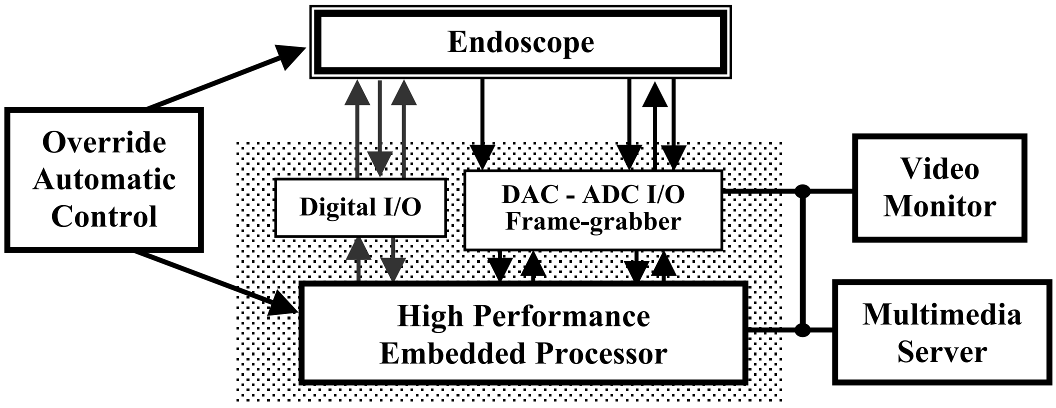

Endoscope
Override Automatic Control
Digital I/O
DAC - ADC I/O Frame-grabber
Video Monitor
High Performance Embedded Processor
Multimedia Server

Figure 1: Endoscope Control Embedded System

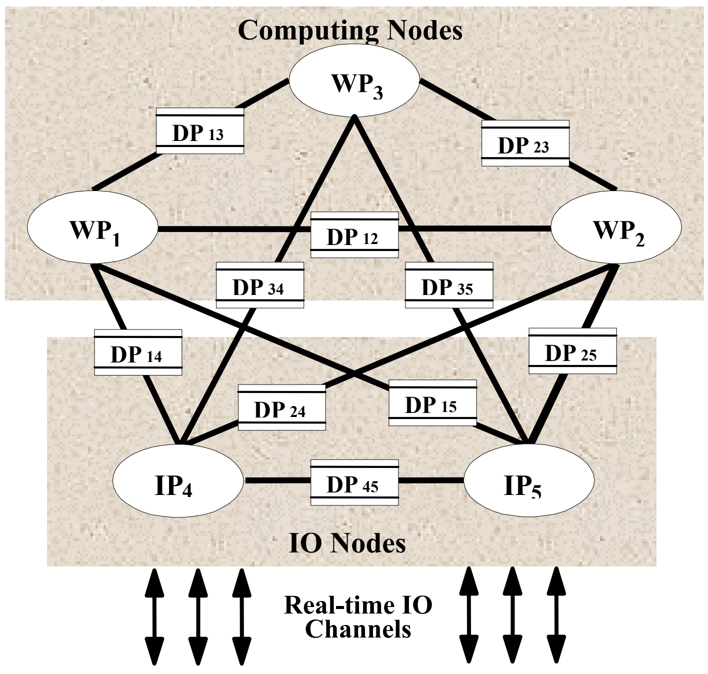


Figure 2. Embedded Processor Architecture

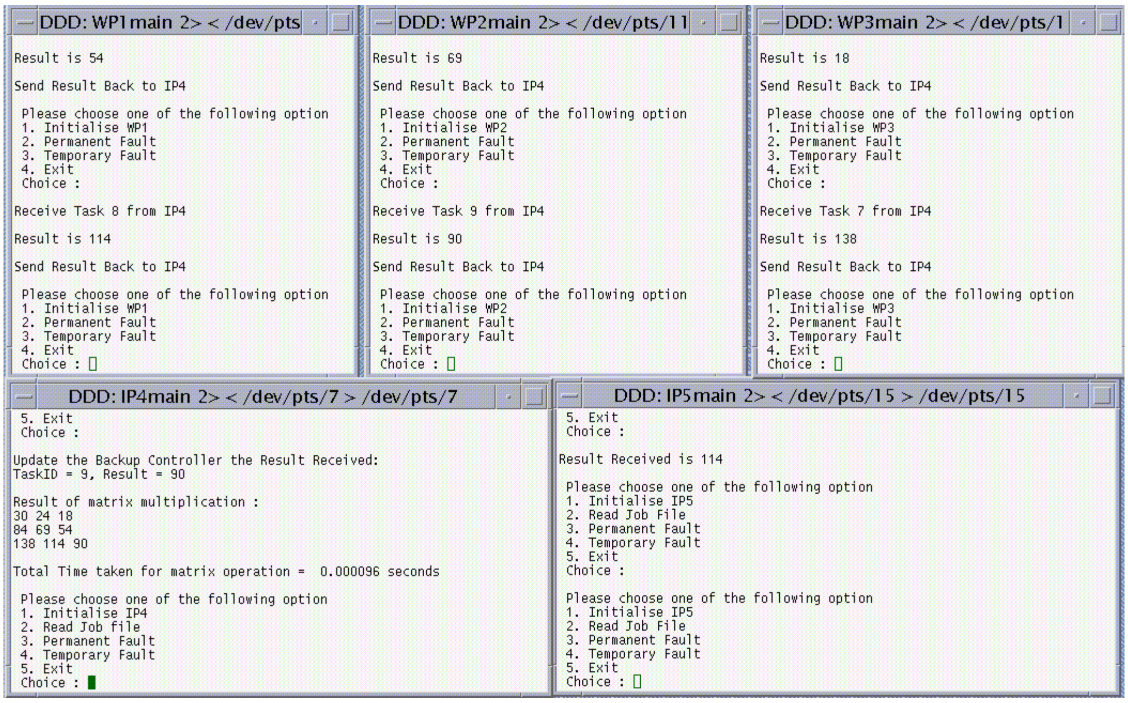
3. EMBEDDED SYSTEM DESIGN

Embedded system design is the best candidate for hardware-software codesign [2]. The codesign approach speeds up the intuitively serial design process by developing hardware and software concurrently [3]. A hardware/software co-verification tool set Eagle*i* is employed that provides a virtual environment for integrating hardware and software subsystems. The software execution environment interacts with the VHDL based hardware model and Virtual Software Processor (VSP) using Eagle*i* Toolset. This has allowed us to test and verify the complete (hardware as well as software) embedded system before a physical SOC (system on a chip) is built. The VSP allows the fetch and execute cycle to be handled by the software application running at the workstation speed, thereby freeing the HDL simulator to handle rest of the system hardware on demand from the software application. Two types of virtual software processor (VSP) are used, Oak DSP for computing nodes and ARM7TDMi for IO nodes. Address decoding, watchdog timer and DP memory glue logic is implemented for VSP interconnections. These hardware modules are developed using VHDL.

A fault-tolerant computer system should have a mechanism for system monitoring and to keep the status of its components in such a way that the status information is available to all the processing nodes. For the embedded system presented in this paper, system status is kept at each node in a 4 x 5 size 2D status register. The status of each node is kept in four fields. The “available” field when set indicates the availability of a particular node. The “busy” field when set indicates a busy node and the node will not accept any new task. The “reset” field when set implies that the node is currently being diagnosed after a failure. The “com-module” field indicates the availability of the communication module between the current node and other nodes. The “TaskID” field is the fifth field that is only kept at system controller (**IP**) nodes. It indicates the task IDs being executed by all the system nodes.

3. HARDWARE SOFTWARE COVERIFICATION

A set of endoscope application algorithms is chosen to verify the fault-tolerance feature of the embedded system architecture. The application is partitioned into smaller tasks to be executed by the compute processors. The system and application software processes are developed in C language. The kernel software is implemented at each node to provide support for system level tasks. Compute nodes execute the main computational intensive components of machine vision and navigation algorithms as well as inter-processor communication processes. One of the **IP** nodes serves as system controller supporting for fault-tolerance and system recovery. It also provides task scheduling, load balancing, real-time control and I/O. All the system nodes execute a fault-injection process for fault simulation purposes.


DDD: WP1main 2> < /dev/pts
DDD: WP2main 2> < /dev/pts/11
DDD: WP3main 2> < /dev/pts/1
DDD: IP4main 2> < /dev/pts/7 > /dev/pts/7
DDD: IP5main 2> < /dev/pts/15 > /dev/pts/15

```
Result is 54

Send Result Back to IP4

 Please choose one of the following option
 1. Initialise WP1
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

Receive Task 8 from IP4

Result is 114

Send Result Back to IP4

 Please choose one of the following option
 1. Initialise WP1
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :
```

```
Result is 69

Send Result Back to IP4

 Please choose one of the following option
 1. Initialise WP2
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

Receive Task 9 from IP4

Result is 90

Send Result Back to IP4

 Please choose one of the following option
 1. Initialise WP2
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :
```

```
Result is 18

Send Result Back to IP4

 Please choose one of the following option
 1. Initialise WP3
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

Receive Task 7 from IP4

Result is 138

Send Result Back to IP4

 Please choose one of the following option
 1. Initialise WP3
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :
```

```
 5. Exit
 Choice :

Update the Backup Controller the Result Received:
TaskID = 9, Result = 90

Result of matrix multiplication :
30 24 18
84 69 54
138 114 90

Total Time taken for matrix operation =  0.000096 seconds

 Please choose one of the following option
 1. Initialise IP4
 2. Read Job file
 3. Permanent Fault
 4. Temporary Fault
 5. Exit
 Choice :
```

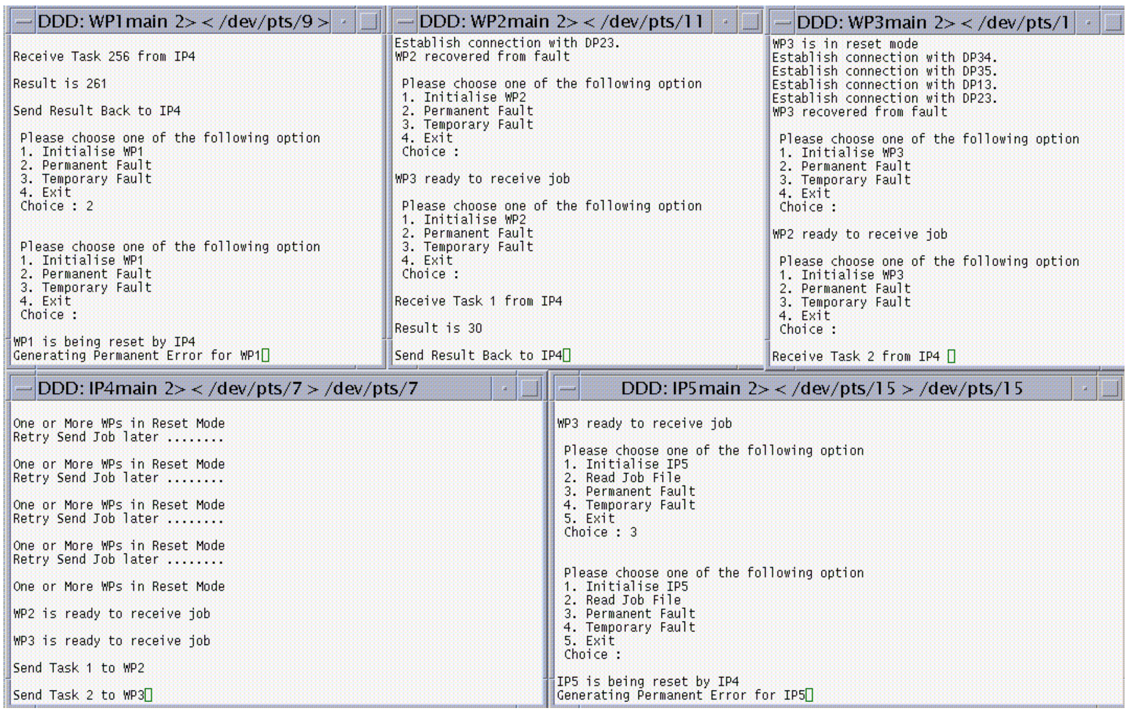
```
 5. Exit
 Choice :

Result Received is 114

 Please choose one of the following option
 1. Initialise IP5
 2. Read Job File
 3. Permanent Fault
 4. Temporary Fault
 5. Exit
 Choice :

 Please choose one of the following option
 1. Initialise IP5
 2. Read Job File
 3. Permanent Fault
 4. Temporary Fault
 5. Exit
 Choice :
```

Figure 3: Normal Operation of Embedded System

A number of hardware and application software faults are simulated and the software-hardware mechanisms for fault-detection, containment and system recovery are verified. All the system nodes check themselves and their interface **DP** memories when the system starts functioning. The system status tables are also initialized at each node. One of the healthy **IP** nodes (**IP4**) becomes the system controller while the other takes over the role of backup controller (**IP5**). A normal operation of the 5-node system is shown in Figure 3 where the upper three windows represent three computing nodes while bottom row windows shows the activities of **IP** nodes. **IP4** distributes the computing tasks to other nodes, which returns the results to **IP4**. We also present a selection of co-verification results for various failure scenarios. Figure 4 presents a scenario where WP1 has failed permanently while **WP2** and **WP3** nodes are injected with a temporary fault. It also shows the system recovery process. We present another failure scenario that shows a temporary fault in IP4 that is serving as system controller. Figure 5 shows the recovery of IP4 from a temporary fault where IP5 takes over as system controller while IP4 becomes backup controller after recovering from failure.

4. CONCLUSIONS

The embedded system processor architecture consists of dual-port memories, micro-controller, fixed-point processor cores and other glue logic. The embedded system is modeled using HDL and Eagle*i* toolset that provides an environment and virtual software processors to build a multiple processor virtual hardware. The system and application software modules are developed and co-verified by using the virtual hardware. Hardware software co-verification results, presented in this paper, indicate that the system degrades gracefully under various fault scenarios. A realistic reliability model of the system would require only two (computing and IO node each) fault-free communicating nodes for the system to be considered operational. In the extreme case, even one healthy **IP** node keeps the embedded system operating at the lowest level of performance. The main objective of this research is to design and develop an SOC (system on a chip) level high performance embedded computer. A VLSI implementation of the embedded processor system is being undertaken.



DDD: WP1main 2> < /dev/pts/9 >

```
Receive Task 256 from IP4

Result is 261

Send Result Back to IP4

 Please choose one of the following option
 1. Initialise WP1
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice : 2


 Please choose one of the following option
 1. Initialise WP1
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

WP1 is being reset by IP4
Generating Permanent Error for WP1
```

DDD: WP2main 2> < /dev/pts/11

```
Establish connection with DP23.
WP2 recovered from fault

 Please choose one of the following option
 1. Initialise WP2
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

WP3 ready to receive job

 Please choose one of the following option
 1. Initialise WP2
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

Receive Task 1 from IP4

Result is 30

Send Result Back to IP4
```

DDD: WP3main 2> < /dev/pts/1

```
WP3 is in reset mode
Establish connection with DP34.
Establish connection with DP35.
Establish connection with DP13.
Establish connection with DP23.
WP3 recovered from fault

 Please choose one of the following option
 1. Initialise WP3
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

WP2 ready to receive job

 Please choose one of the following option
 1. Initialise WP3
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

Receive Task 2 from IP4
```

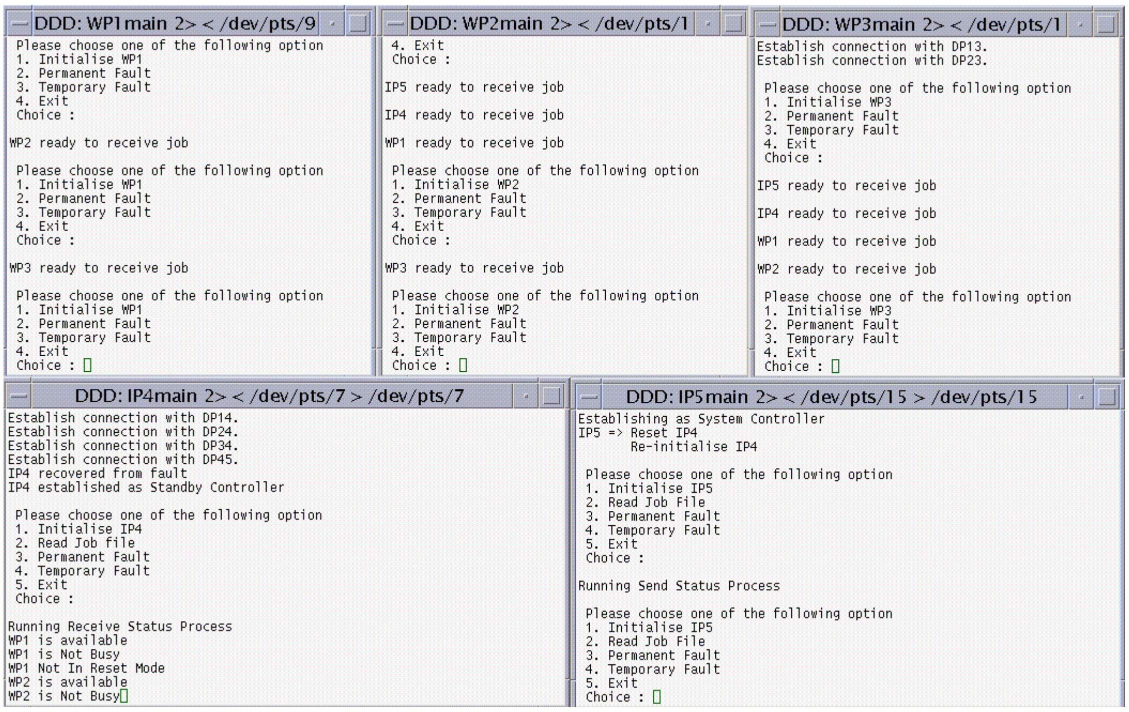
DDD: IP4main 2> < /dev/pts/7 > /dev/pts/7

```
One or More WPs in Reset Mode
Retry Send Job later ........

One or More WPs in Reset Mode
Retry Send Job later ........

One or More WPs in Reset Mode
Retry Send Job later ........

One or More WPs in Reset Mode
Retry Send Job later ........

One or More WPs in Reset Mode

WP2 is ready to receive job

WP3 is ready to receive job

Send Task 1 to WP2

Send Task 2 to WP3
```

DDD: IP5main 2> < /dev/pts/15 > /dev/pts/15

```
WP3 ready to receive job

 Please choose one of the following option
 1. Initialise IP5
 2. Read Job File
 3. Permanent Fault
 4. Temporary Fault
 5. Exit
 Choice : 3


 Please choose one of the following option
 1. Initialise IP5
 2. Read Job File
 3. Permanent Fault
 4. Temporary Fault
 5. Exit
 Choice :

IP5 is being reset by IP4
Generating Permanent Error for IP5
```

Figure 4 : System Recovery from a Permanent Fault at WP1 and Temporary Failure of WP2 and WP3



DDD: WP1main 2> < /dev/pts/9

```
 Please choose one of the following option
 1. Initialise WP1
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

WP2 ready to receive job

 Please choose one of the following option
 1. Initialise WP1
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

WP3 ready to receive job

 Please choose one of the following option
 1. Initialise WP1
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :
```

DDD: WP2main 2> < /dev/pts/1

```
 4. Exit
 Choice :

IP5 ready to receive job

IP4 ready to receive job

WP1 ready to receive job

 Please choose one of the following option
 1. Initialise WP2
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

WP3 ready to receive job

 Please choose one of the following option
 1. Initialise WP2
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :
```

DDD: WP3main 2> < /dev/pts/1

```
Establish connection with DP13.
Establish connection with DP23.

 Please choose one of the following option
 1. Initialise WP3
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :

IP5 ready to receive job

IP4 ready to receive job

WP1 ready to receive job

WP2 ready to receive job

 Please choose one of the following option
 1. Initialise WP3
 2. Permanent Fault
 3. Temporary Fault
 4. Exit
 Choice :
```

DDD: IP4main 2> < /dev/pts/7 > /dev/pts/7

```
Establish connection with DP14.
Establish connection with DP24.
Establish connection with DP34.
Establish connection with DP45.
IP4 recovered from fault
IP4 established as Standby Controller

 Please choose one of the following option
 1. Initialise IP4
 2. Read Job file
 3. Permanent Fault
 4. Temporary Fault
 5. Exit
 Choice :

Running Receive Status Process
WP1 is available
WP1 is Not Busy
WP1 Not In Reset Mode
WP2 is available
WP2 is Not Busy
```

DDD: IP5main 2> < /dev/pts/15 > /dev/pts/15

```
Establishing as System Controller
IP5 => Reset IP4
       Re-initialise IP4

 Please choose one of the following option
 1. Initialise IP5
 2. Read Job File
 3. Permanent Fault
 4. Temporary Fault
 5. Exit
 Choice :

Running Send Status Process

 Please choose one of the following option
 1. Initialise IP5
 2. Read Job File
 3. Permanent Fault
 4. Temporary Fault
 5. Exit
 Choice :
```

Figure 5: Temporary Failure and Recovery of IP4

References

1. Gul N. Khan and Duncan F. Gillies, "Vision Based Navigation System for an Endoscope, "*Image and Vision Computing,* Vol. 14, No. 10, pp. 763-772, 1996.
2. M. Chiodo, P. Giusto, A. Jurecska, H. C. Hsieb, A. S. Vincentelli and L. Lavagno, "Hardware-Software Codesign of Embedded Systems", *IEEE Micro*, Vol. 14. No. 4, pp. 26-36, August 1994.
3. F. Slomka, M. Dorfel, R. Munzenberger and R. Hofmann, "Hardware/Software Codesign and Rapid Prototyping of Embedded Systems", *IEEE-Design and Test of Computers*, Vol.17, No.2, pp.28-38, April-June 2000.